Complete Contingency Planners

Dennis Kibler
kibler@ics.uci.edu
Karl Schwamb
schwamb@ics.uci.edu

Technical Report 92–24

February 26, 1992

This research was supported by National Science Foundation grant number IRI-9001756.

Complete Contingency Planners

Dennis Kibler
kibler@ics.uci.edu

Karl Schwamb
schwamb@ics.uci.edu

Dept. of Info. & Computer Science
University of California, Irvine
Irvine, CA 92717-3425 U.S.A.
(714) 856-5951

Keywords: Planning, uncertainty.

Abstract

A framework is proposed for the investigation of planning systems that must deal with bounded uncertainty. A definition of this new class of *contingency* planners is given. A general, complete contingency planning algorithm is described. The algorithm is suitable to many incomplete information games as well as planning situations where the initial state is only partially known. A rich domain is identified for the application and evaluation of contingency planners. Preliminary results from applying our complete contingency planner to a portion of this domain are encouraging and match expert level performance.

1. Introduction

Traditional AI planning approaches utilize simplified models of the world in which all information needed to plan is known and all actions are performed solely by the planning agent. While these models stimulated research for a number of years, it is increasingly clear that they ignore important issues critical to real-world applications. Dissatisfaction with this approach has triggered the development of alternatives such as "reactive" planning systems (Brooks, 1986; Agre & Chapman, 1987). Planning research in this area centers on time-critical decision-making strategies due to a rapidly changing or highly complicated environment with numerous interacting agents.

While our research is also concerned with interacting agents, we believe *explicit* reasoning about uncertainties in complex worlds is common. Consequently, planning techniques developed to deal with such considerations have a potentially wide range of application. Rather than limit ourselves to extremely short deadlines for decision making, we focus on worlds in which there is time for deliberation before acting. Furthermore, we represent uncertainties symbolically to *guarantee* goal satisfaction.

2. Contingency Planning

Most of our plans in daily life are contingency plans. Consider a simple plan for driving to the store. Most of us have a spare tire in case of a flat. Some of us have insurance coverage in case we have a more serious problem. A few of us even have a cellular phone in case there is no phone nearby. But whatever plan we formulate for a contingency, its success depends on certain assumptions about the world. Let us now consider a formal development of the notion of a contingency plan and its associated assumptions.

When other agents are present in the planning environment, a game-like situation is created. Since other agents can apply operators of their own, a straightforward path to a goal state might not be achievable by the planning agent. The alternation of planning agent steps with the steps of other agents is naturally represented using a tree and is termed a *game tree.* A game tree which contains all possible steps by both the planning agent and other agents but whose leaves consist only of goal states is named a solution tree. The solution tree defines the space of all possible solutions which begin at the initial state. A *single-world contingency plan* begins at a single initial state (i.e., the plan's world) and forms a subgraph of the solution tree in which

only one planning agent step emanates from any given state. The leaves of the single-world contingency plan are a subset of the leaves in the solution tree, each of which terminate a path through the solution tree beginning at the initial state. Furthermore, the contingency plan guarantees that starting at the initial state, a goal state can be reached.

As developed thus far, a single-world contingency plan can represent uncertainty in the actions of other agents. The same structure also can be used to represent a second form of uncertainty: uncertainty in the result of operator application. This second form of uncertainty implies that several states may arise after a step, not just one. This can be represented by a perfect operator application (i.e., one that moves to a single, well-defined state), followed by the action of an artificial agent. This artificial agent applies some other step to alter the basic action of the planning agent's operator. This could range from some small random perturbation to a systematic adjustment of the intended action of the step. Because this is handled by the same mechanism used for the actions of other agents, we do not elaborate further on this form of uncertainty.

A third form of uncertainty arises from the fact that the initial state may be only partially known – a common characteristic of real-world planning. We model this by using a new type of state to represent sets of states in the domain. More precisely, we define $\mathcal{S}$, a multistate, to be a *set* of states. The definition of operator application is extended to a multistate in the natural way, i.e., $o\mathcal{S} = \{os : s \in \mathcal{S} \text{ and } s \text{ satisfies the preconditions of operator } o\}$. In a contingency plan we alternate planning agent actions with the actions of other agents. A planning agent operator may be applied only when it is applicable to every state in the multistate. Otherwise, one of the initial states might lead to a "dead-end". The operator of another agent is applicable when it applies to at least one of the states in the multistate.

Each multistate, $\mathcal{S}_i$, is viewed as a node in a generalized contingency plan tree. $\mathcal{S}_0$ defines the root of the tree. When every state of a node $\mathcal{S}_i$ is a goal state, then the node $\mathcal{S}_i$ is identified as a goal node. Hence, a *multi-world contingency plan*, or simply *contingency plan*, is the tree consisting of these nodes connected by their associated operator applications. The root node $\mathcal{S}_0$ identifies the possible worlds in which the plan achieves a goal.

The potentially large number of possible initial states, say N, makes contingency planning much more difficult than ordinary planning. While the multistate formulation enables standard game tree search techniques to gen-

erate contingency plans, the approach is costly. Since a contingency plan must indicate the initial states to which it applies, a planner might potentially search a space of $2^N - 1$ combinations of initial states. In general, the requirement of dealing with N initial states increases the search space for a plan by a factor of 2^N. In the next section, we develop a more efficient approach for generating contingency plans based on minimax search.

The contingency plan which works for the largest set of initial states covers the largest set of contingencies. This identifies a preferable plan. A contingency plan is *maximal* if its initial states are not a subset of the initial states associated with any other contingency plan.

This formulation is considerably more general than it may appear. First, the use of strictly alternating moves is not critical to the development of a contingency plan. The planning agent or other agents could make several steps in sequence leading to the planning agent moves occurring on non-alternating plies. Since the selection of moves and the propagation of their consequences is done locally, the proper contingency plan structure would be created. This approach would be useful in environments where moves cannot be synchronized. Second, the use of a set of initial states as defining the possible worlds of existence is simply a normal form for the representation of assumptions. Individual assumptions define a particular characteristic of the world and a set of assumptions define a possible world (Doyle, 1979; de Kleer, 1986). Since assumptions are antecedent to any inference chain, any set of assumptions may be viewed as completing the specification of an otherwise incomplete initial state.

Using the above notion of a contingency plan, we now present a framework for the development of planning systems which produce such plans. A contingency planner's inputs include those of traditional planners: an initial state, a set of operators for the planning agent, and a goal predicate. Uncertainties are represented through the inclusion of an additional set of operators for the other agents and a finite set of possible worlds that serve to characterize the uncertainties involved in the initial state.

The planner's objective is to produce a set of plans which satisfy the desired goal predicate. A set of possible worlds is associated with each plan and serves to define the conditions under which the plan will satisfy the goal. Generally, a set of plans is produced since different plans may be appropriate to different possible worlds. The output set of plans is a maximal set as defined above. Figure 1 summarizes this contingency planner specification.

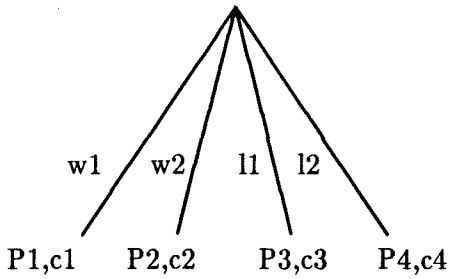
Input:	A finite set of initial states A finite set of operators for the planning agent A finite set of operators for other agents A goal predicate
Output:	A finite set of plan/possible worlds pairs such that: - *Each plan is guaranteed to work if one of its possible worlds holds* - *No plan's possible worlds is a subset of those of any other plan*

Figure 1: Contingency planner specification

3. A Complete Contingency Planner

The overall objective in our planning approach is to find a set of plans and a set of possible worlds under which the plans hold. For each plan, these possible worlds are the weakest preconditions under which the plan will succeed. For expository purposes, we first describe an alternate form of the contingency planning algorithm. Under conditions of *certainty*, plans for achieving a goal in the presence of an adversary can be easily generated by applying the minimax algorithm. We note that a plan is formed by the nodes traversed in the search process. A plan would consist only of "max" moves at the planning agent's turn and *all* possible moves at the other agents' turn. Since all possible moves are retained for the other agents, the representation can account for those agents which may be benevolent or indifferent in addition to antagonistic. Since there can be several "max" moves for the planning agent, a set of plans is generated.

The uncertainties considered in our framework are finite and define a space of possible worlds. For each possible world, the minimax procedure can be applied to produce a set of plans. Hence, a set of <possible world>/<plan> pairs can be computed, one pair for each plan generated from the minimax procedure for the given possible world. Note that the same plan can be generated for different possible worlds. From these pairs, a maximal set can be formed by considering a partial order over sets of possible worlds. A plan P_1 subsumes another, P_2, if and only if $W_1 \supset W_2$, where $W_i = \{w : w/P_i$ holds$\}$. This forms a partial order on the set of plans, and the maximal set is formed by simply taking the maximal elements of the corresponding lattice.



Offense: Choice(w1:P1,c1; w2:P2,c2)

Defense: Forbid(l1,l2) ∩ [c1∩c2 ∪ Forbid(w1)∩c2 ∪ Forbid(w2)∩c1]

Figure 2: A single step subtree and corresponding constraints

The final result is a maximal set of plans, each of which has an associated set of possible worlds.

Our actual algorithm, termed the μ algorithm, is essentially an optimization of the above process. Plans are computed incrementally and in parallel during the minimax search process. The possible worlds associated with a plan are maintained for each plan as it is being updated. After each update, a subsumption calculation is used to keep the size of sets small.

The remainder of this section describes this more elaborate process used by our implementation. We adopt game theory terminology to concisely describe the algorithm: achieving a goal is termed a *win*, operator applications are termed *plays*, the planning agent is the *offense*, and the other agents are the *defense*. We consider a simplified case first. Figure 2 shows an abstract form of a single plan step, with four possible operators, leading to two wins (w1,w2) and two losses (l1,l2). Each leaf is a single plan (e.g., P1) and its associated constraint (possible world set). Let's consider how constraints are formed for both offensive and defensive play. The new plan(s) created at the node (the root in Figure 2) are formed bottom-up from its children. The offensive case is easiest: simply form a *choice* expression with the winning plays. Each separate play represents a winning plan. The defensive case is more complex. To win, we must first exclude the possibility of playing losing lines. This produces a set of possible worlds which are winnable since defense cannot select lines which lose for the offense. To complete the constraint expression, we must consider winning possibilities in turn. First, a set of worlds satisfying all constraints may exist, so one possibility is found by intersecting all the constraints of winning plays. Another possibility is that a set of worlds may exist which enable one of the winning plays to be made. We find this set by excluding the other winning play and intersecting

that set of worlds with the set of worlds implied by the constraints of the winning play. This produces the expression for defensive play in Figure 2.

Finally, consider how constraints are formulated in the general case. The offensive case is trivial and simply produces a choice constraint resulting in one plan for each possible winning play. The remainder of the discussion focuses on the defensive case.

In general, the play from a node results in some wins (with constraints) and some losses. Let M be the set of all possible moves from the node. Let W be the set of moves leading to a win. Let L be the set of moves leading to a loss. Clearly, W and L disjointly partition M. To find the constraint applicable to the node we must consider several factors. Since all the losing situations must be avoided, we must forbid all $l \in L$ from occurring. This generates one of the general constraints.

Next we consider the plays W which can lead to a win. For each $w \in W$, let c_w be the constraint which guarantees winning. One simple case occurs if a world satisfies every c_w for all $w \in W$. For such a world, winning is guaranteed. There are more complicated situations in which winning is also guaranteed, such as when a world satisfies only one c_w. In that case, we must add the constraint that the other plays in W are forbidden. Let S be an arbitrary subset of W. The *contingency constraint formula* is a general expression which defines the constraints for a node:

$$forbid(L) \cap (\bigcup_{S \subseteq W} forbid(\tilde{S}) \cap S_c)$$

where $forbid(X)$ is the set of worlds in which no $x \in X$ can be played, $\tilde{S}$ is the complement of S in W (i.e., $W - S$), and S_c is $\bigcap_{w \in S} c_w$ where c_w is the constraint associated with w. This formula generates three types of pruning during the search for contingency plans. If a constraint cannot be satisfied it is eliminated. If a constraint is incompatible with another then their intersection is empty and the two possible constraints are eliminated. Finally, if one constraint applies to a strict subset of another then the elements in the difference are eliminated, since only those which work for all contingencies are retained. We illustrate the use of this algorithm in the following section where a concrete domain for evaluation is identified.

4. A Domain for Contingency Planning

To stimulate the further development and communication of contingency planning ideas we have searched for a readily accessible domain. To evaluate the techniques developed for contingency planning we have searched for a domain complex enough to be challenging yet simple enough to yield small testable examples. These qualities are presented in the domain of *contract bridge* play. In particular, the aspect of the game involving card play has proved to be a fertile testing ground for contingency planning.

Bridge offers a domain with bounded uncertainty (regarding card placement), adversary planning, and multiple agents. While a full analysis of any particular hand in bridge would involve an intractable 52-play lookahead, there are known experts in the domain. The domain's tractability, at least for people, is further underscored by the fact that even beginning tournament players, who number in the hundreds of thousands, generate plans for an entire hand in the allotted time of 2-3 minutes. Post mortem analyses often revolve around the identification of reasonable planning assumptions. There are over 5 million $\binom{25}{12}$ possible starting defensive distributions (i.e., starting states) for any given hand. Of course, since the contingency planning agent is attempting to develop plans which work for a *set* of distributions, the space of possible worlds to consider is $2^{5000000}$. Even from a single initial state, i.e., one defensive card distribution, a search space is defined on the order of 2.6×10^{10} states[1].

Another reason for choosing this domain is the existence of numerous texts containing problems of graded difficulty which present a good basis for performance evaluation. Instruction and play have evolved to the point that even complicated plans can be communicated in a few English sentences.

Bridge is a four player game with players known as North, South, East, and West, corresponding to their placement around a card table. North and South are partners playing against East and West. In each round, an offensive player plays the first card (the lead) and the other players play one card in turn with the highest card winning.

To illustrate the μ algorithm in this domain, consider how two elementary

[1]The first author, in conjunction with Dan Hirschberg, has developed previously a program to analyze double dummy hands using a combination of efficient search methods. This program exhibits a branching factor of 1.6. Hence, after the opening lead, a search space is defined on the order of $1.6^{51} \approx 2.6 \times 10^{10}$.

North:	♣ A Q
South:	♣ x x
Defense:	♣ K x x x
Goal:	Two tricks

Figure 3: Example two-card suit problem

plans are generated in a 2-card suit combination. North holds ♣ A Q opposite South's ♣ x x, where "x" denotes a small card. The defense holds the cards ♣ K x x x in an unknown distribution. We assume that South has the lead and the goal of taking two tricks. The problem is summarized in Figure 3. The algorithm finds plans for taking two tricks and the associated card distributions in which the plans work. Figure 4 illustrates the possible card plays beginning with South's lead.

After South leads ♣ x, West might play ♣ K or ♣ x. If West has no cards in the suit, i.e., is void, then a card from some other suit must be played. This is designated by "V". This is followed by North's possible plays, and then East's. The last round is forced, so we simply show the result at the leaves of the tree: W for a win of two tricks, L for achieving less than two tricks. The nodes of the first three ply are annotated with numbers indicating the order in which plans are composed as well as the constraints under which a win is possible. Plans are composed bottom-up. Clearly at node 1 (indicated by $\boxed{1}$) an unconditional win is indicated since all its children lead to wins. Similarly, node 2 indicates an unconditional loss. Hence, at node 3, an unconditional win is indicated since the choice of moving to node 1 or 2 is an offensive choice and the planner will select the winning play.

At node 4, a win can be obtained only when the defense plays ♣ K. Since this is a defensive choice, this will only happen when the other possible plays cannot be made. This is the case when East has only the king. Therefore, at node 4 a win is generated under the condition that East has the singleton king (shown in Figure 4 as East=K). This determination is made at this node, and all others, using the contingency constraint formula: ♣ x and void are excluded from any possible distribution for East since they are losses ($forbid(L)$) and the result is intersected with the distribution associated with the winning ♣ K play (S_c). The remaining term in the formula vanishes since there is only one win. Similarly, at node 5, a win is generated when East does not have ♣ K (shown as East-K). At node 6, we generate a plan by

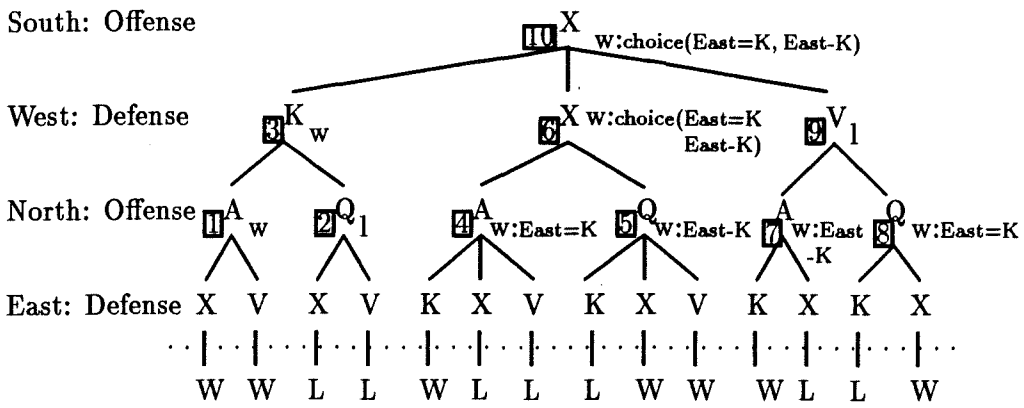


Figure 4: Generation of finesse and drop plans

considering the plans of the children (as was done for node 3). Since either play (♣ A or ♣ Q) leads to a win (but under different conditions) we create two plans and indicate this choice. Following this same reasoning, it can be determined that node 9 leads to an unconditional loss.

The final set of plans is generated at node 10. Since the defensive play of ♣ K at node 3 leads to an unconditional win, this places no constraints on the possible worlds for which any of the plans will work. Node 9, on the other hand, is an unconditional loss, so the single distribution where West has no clubs is excluded (i.e., no winning plan for that case can be found). Two cases are generated from the constraints at node 6, the same as discussed above. For each of the two cases, an associated plan tree is generated by retaining the winning offensive card plays associated with the cases. The case where East has the singleton king yields a plan including the nodes 6, 4, and the ♣ K child of node 4. The other case, where East does not have ♣ K, includes nodes 3, 1 (and its descendants), 6, 5, and the ♣ X and void children of node 5. The first plan (for the case where East has ♣ K only) is known as a *drop* plan. The other is known as a *finesse*. Both are summarized in the following table:

Name	Possible Worlds	Plan Summary
Drop	East has ♣ K	South plays low to ♣ A. Next ♣ Q.
Finesse	West has ♣ K and possibly some small cards	South plays low to ♣ Q, if West plays ♣ K, play ♣ A. if West plays low, play ♣ Q

5. Evaluation

Suit combination planning in the bridge domain has produced strong results. The complete contingency planning algorithm is able to solve a range of single suit card combination problems. It is able to form plans for the elementary finesse and drop plans taught to beginning bridge players.

More impressive are the planner's solutions to a set of problems given in a recent bridge text for players of moderate to high skill. Lawrence (1988) spends an entire book discussing the playing technique for seven difficult card combinations. The μ algorithm is able to generate correct plans for each of these hands, demonstrating expert level performance in this subproblem task. These problems are close to the most difficult problems possible in this domain. On the most difficult card combinations, the algorithm searches through a space of $2^{2^6} \approx 1.8 \times 10^{19}$ possible world sets to find the solutions.

One of the tricky problems in (Lawrence, 1988) is North: ♠ J 10 x opposite South: ♠ A x x. Defense has the remaining cards, ♠ K Q x x x x x, in some unknown distribution. The goal is to obtain two tricks. The algorithm finds 3 plans which are summarized using bridge play terminology in Figure 6. Each plan is given along with the possible worlds under which they work. For example, the second plan works in the 6 possible worlds listed.

The performance of the μ algorithm on the entire Lawrence problem set is illustrated in Figure 7. The first three columns of the figure list the problem number from (Lawrence, 1988),

Possible Worlds	**Plan Summary**
West has ♠ K or ♠ Q West has ♠ K x or ♠ Q x East has ♠ K Q	South plays low to ♠ J, if West plays honor, duck and finesse East if West plays low, play ♠ J and next ♠ A
West has ♠ K or ♠ Q East has ♠ K or ♠ Q West has ♠ K Q East has ♠ K Q	South plays the ♠ A, next round North plays the ♠ J
East has ♠ K or ♠ Q East has ♠ K x or ♠ Q x West has ♠ K Q	Lead ♠ J and cover West's play, next lead ♠ A

Figure 6: Solution to North: ♠ J 10 x opposite South: ♠ A x x

#	PWs	Depth	G Nodes	μ Nodes	Δ
1	24	12	4.7×10^9	1.6×10^4	10^5
2	32	8	1.8×10^{11}	1.9×10^3	10^8
3	32	12	1.2×10^{12}	4.2×10^4	10^8
4	16	16	1.2×10^8	4.7×10^4	10^4
5	12	16	7.6×10^6	4.6×10^3	10^3
6	40	12	3.1×10^{14}	2.4×10^4	10^{10}
7	40	12	3.1×10^{14}	8.0×10^4	10^{10}

Figure 7: Performance of μ algorithm.

the number of possible worlds (PW) associated with each problem, and the depth of the plan trees generated. The fourth column gives an estimate of the number of nodes explored using the generate and test (G) approach to contingency planning mentioned in section 2. That approach selects a set of possible worlds then generates a plan using the multistate representation, repeating the process over all sets of possible worlds. The number of nodes explored by that approach would be $O(2^{PW} b^{depth})$, excluding the calculation for maximal elements. In this domain the branching factor, b, is approximately 1.6, as noted earlier. The last two columns give the actual number of nodes explored by the μ algorithm on the same problems along with the resulting improvement over the brute force method.

6. Conclusions

We have presented a new theory of planning for contingencies. We have outlined a framework for the characterization of contingency planners, including a definition of a contingency planner. We have developed the first incomplete-information planning algorithm, the μ algorithm. This algorithm has achieved expert level performance in the domain of bridge, more specifically, the area of card combination play. By taking advantage of constraint violations, the μ algorithm greatly reduces the amount of search done over a naive approach. Currently, we are unable to theoretically predict this reduction as it depends on the particular constraints that arise. Nevertheless, the approach is efficient and further work is under way to extend the approach to problems on a larger scale. An approach patterned after (Ruby & Kibler, 1991) is under development and involves problem decomposition, planning contingencies for subproblems, and the merging of subproblem solutions.

References

Agre, P., & Chapman, D. (1987). Pengi: An implementation of a theory of activity. *Proceedings of the Sixth National Conference on Artificial Intelligence.* San Mateo, CA: Morgan Kaufmann.

Brooks, R. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation, 2*, 1, 14–23.

de Kleer, J. (1986). An assumption-based truth maintenance system. *Artificial Intelligence, 28*, 127–162.

Doyle, J. (1979). A truth maintenance system. *Artificial Intelligence, 12.*

Lawrence, M. (1988). *How to play card combinations.* Louisville, KY: Devyn Press.

Ruby, D., & Kibler, D. (1991). SteppingStone: An empirical and analytical evaluation. *Proceedings, Ninth National Conference on Artificial Intelligence.* San Mateo, CA: AAAI Press.